



# **Commande de l'avance d'un train sur la plateforme UniRAIL**

**Armand Toguyéni**  
**Professeur des Universités**  
**Ecole Centrale de Lille**  
**Bur. C341 Tel . 03-20-33-54-49**  
**mel. : [armand.toguyeni@centralelille.fr](mailto:armand.toguyeni@centralelille.fr)**

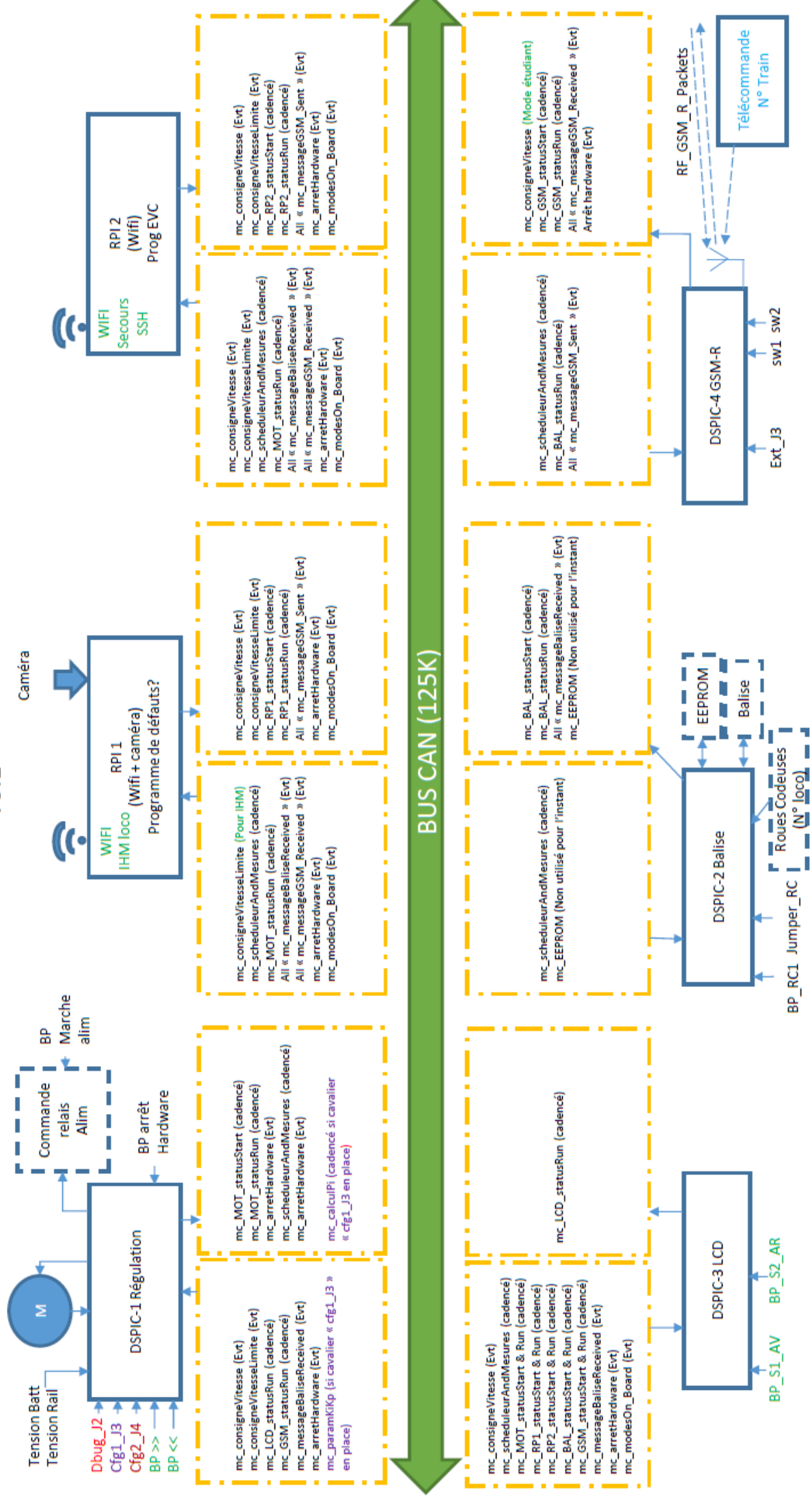


# Plan



- **Bus CAN**
- **Présentation de l'API CAN UniRAIL**
- **Présentation de “canlinux”**
  - ◆ **Ecriture d'une trame CAN**
  - ◆ **Lecture d'une trame CAN**

# V3.1

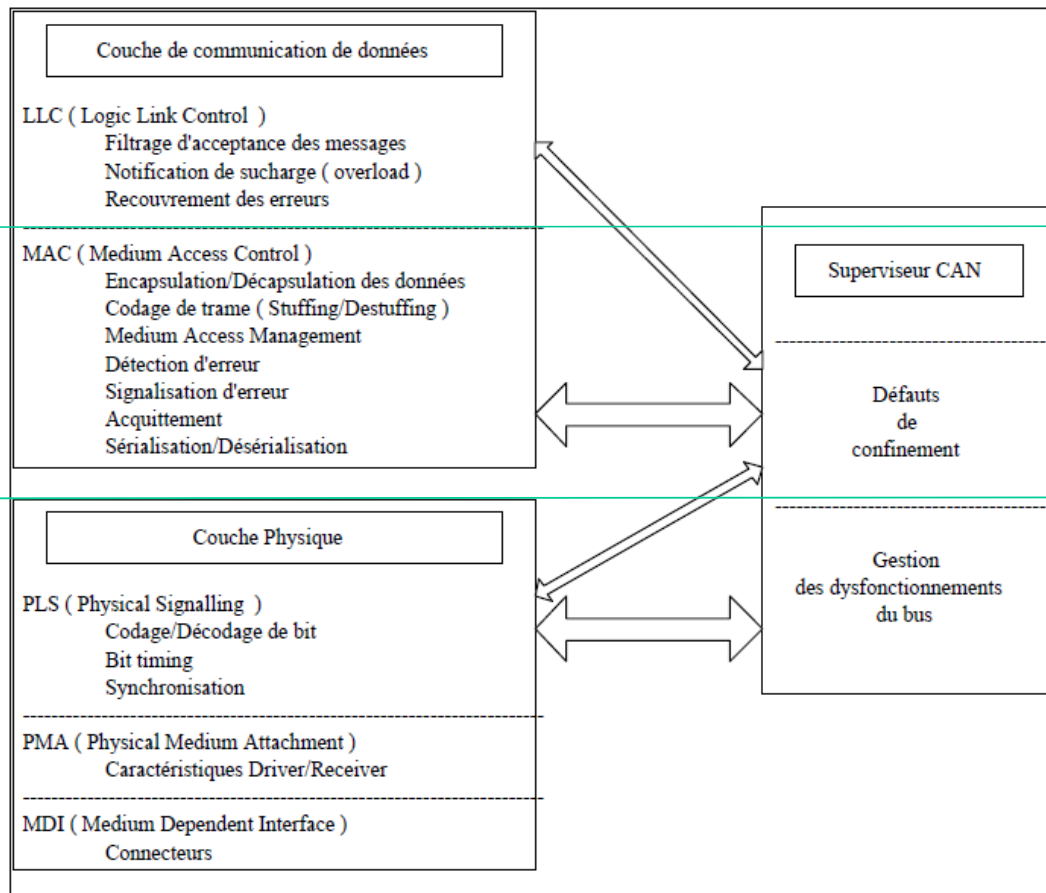


# CAN (1) : historique

- ❑ 1980 : émergence de nombreux systèmes électroniques au sein des véhicules  
⇒ Encombrement et complexité (câblage)
- ❑ 1983 : création du bus CAN par la société Robert Bosch GmbH
- ❑ 1985 : convention avec Intel
- ❑ 1986 : standardisation ISO
- ❑ 1995 : 1<sup>ère</sup> voiture à utiliser le bus CAN
- ❑ 2000 : environ 600 millions de circuits CAN vendus



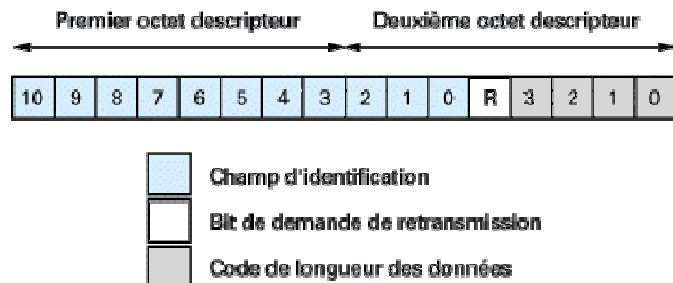
# CAN (2) : Positionnement par rapport au modèle OSI



Méthode d'accès : CSMA/BA

Transmission	Topologie	Longueur	Débit
Diffusion	Bus	50 m. 1000 m.	1 Mbps 50 Kbps

# CAN (3) : format de trame



- *SOF*: Start Of Frame, 1 bit dominant (pour synchronisation)
- *Champ d'arbitrage*:
  - Identificateur (11 bits en format standard, 29 en étendu)
  - RTR : nature de la trame : données (dominant) ou requête (récessif)
- *Champ de contrôle*:
  - 2 bits "en réserve"
  - DLC : 4 bits pour taille des données (*Data LengthCode*)
- *Champ de données* (< 8 octets)
- *Champ de CRC* (15 bits + 1 délimiteur récessif)
- *Champ ack*: 1 bit ack 1 bit délimiteur (récessif)
- *Fin de trame*: 7 bits récessifs sans *bit stuffing*

## CAN (4) : Types de trames

### ❑ Trame de données

- ◆ Diffusion, adressage par id. des données
  - ◆ Modèle producteur/consommateur

### ❑ Trame de requête:

- ◆ Permet de récupérer des infos sur l'objet d'identifiant spécifié dans la trame
  - ◆ Modèle client/serveur

### ❑ Trame d'erreur:

- ◆ Champ erreur : erreur "active" ou "passive«
- ◆ Délimiteur

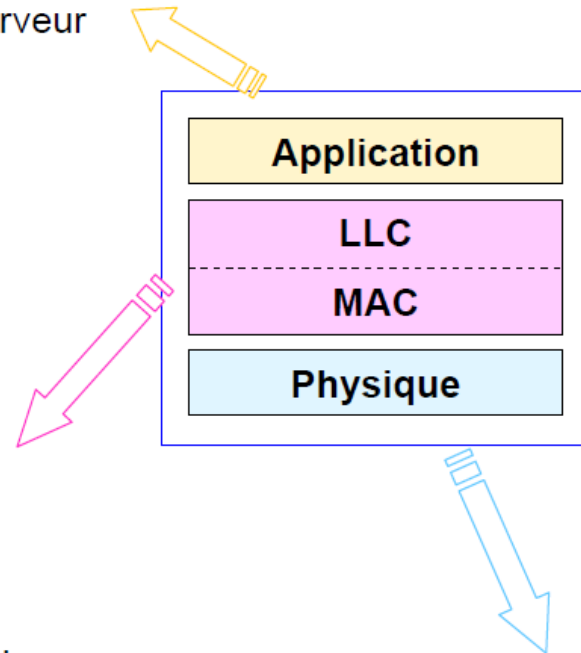
### ❑ Trame de surcharge

- ◆ Pour retarder l'envoi d'une trame
- ◆ Exemple : si un récepteur est saturé.
  - ◆ Overloadflag
  - ◆ Délimiteur

# CAN (5) : Modèle

## ❖ Couche application :

- modèle producteur / consommateur
- modèle client / serveur



## ❖ Couche liaison :




















- arbitrage bit à bit
- gestion erreur réseau

## ❖ Couche physique :

- notion de bits dominant et récessif



# API CAN UniRAIL (1)

Nom	Modifié le	Type	Taille
 canLinux	17/09/2020 10:58	Fichier C	10 Ko
 canLinux	17/09/2020 10:58	Fichier H	5 Ko
 canLinux.h.gch	17/09/2020 10:58	Fichier GCH	2 341 Ko
 loco_Parametres	17/09/2020 10:58	Fichier H	2 Ko
 loco_Parametres.h.gch	17/09/2020 10:58	Fichier GCH	1 921 Ko
 MESCAN1_common	17/09/2020 10:58	Fichier H	1 Ko
 MESCAN1_common.h.gch	17/09/2020 10:58	Fichier GCH	1 921 Ko
 MESCAN1_DefinitionVariable	17/09/2020 10:58	Fichier H	3 Ko
 MESCAN1_DefinitionVariable.h.gch	17/09/2020 10:58	Fichier GCH	8 Ko
 MESCAN1_ID&DLC_INFRA	17/09/2020 10:58	Fichier H	9 Ko
 MESCAN1_ID&DLC_LOCO	17/09/2020 10:58	Fichier H	5 Ko
 MESCAN1_Utilitaire	17/09/2020 10:58	Fichier C	17 Ko
 MESCAN1_Utilitaire	17/09/2020 10:58	Fichier H	16 Ko
 MESCAN1_VarInfraBalAig	17/09/2020 10:58	Fichier C	32 Ko
 MESCAN1_VarInfraBalAig	17/09/2020 10:58	Fichier H	24 Ko
 MESCAN1_VarStatusTrain	17/09/2020 10:58	Fichier C	23 Ko
 MESCAN1_VarStatusTrain	17/09/2020 10:58	Fichier H	19 Ko
 MESCAN1_VarTrain	17/09/2020 10:58	Fichier C	8 Ko
 MESCAN1_VarTrain	17/09/2020 10:58	Fichier H	8 Ko

## API CAN UniRAIL (2)

Bibliothèque : MESCAN1\_DefinitionVariable.h

```
//Type de message CAN
#define CAN1_MSG_DATA      0x01
#define CAN1_MSG_RTR      0x02
#define CAN1_FRAME_EXT    0x03
#define CAN1_FRAME_STD    0x04
#define CAN1_BUF_FULL     0x05
#define CAN1_BUF_EMPTY    0x06
```

```
// Type message CAN
typedef union {
    struct {
        uint32_t id;
        uint8_t idType;
        uint8_t msgtype;
        uint8_t dlc;
        uint8_t data0;
        uint8_t data1;
        uint8_t data2;
        uint8_t data3;
        uint8_t data4;
        uint8_t data5;
        uint8_t data6;
        uint8_t data7;
    } frame;
    unsigned char array[16];
} uCAN1_MSG;
```

## API CAN UniRAIL (3)

□ **Bibliothèque : MESCAN1\_Utilitaire.h**

□ **void MESCAN\_SetBitHigh(uCAN1\_MSG \*CanMsg, MC\_Bit DefBit);**

◆ **Positionne un bit d'une trame CAN à 1.**

◆ **Exemple :**

◆ uCAN1\_MSG consigneVitesse;

◆ MESCAN\_SetBitHigh (&consigneVitesse, bdmc\_sensDeplacementLoco,);

□ **void MESCAN\_SetBitLow(uCAN1\_MSG \*CanMsg, MC\_Bit DefBit);**

◆ **Positionne un bit d'une trame CAN à 0.**

□ **void MESCAN\_SetBit(uCAN1\_MSG \*CanMsg, MC\_Bit DefBit, bool valbit);**

◆ **Positionne un bit à la valeur "valbit".**

## API CAN UniRAIL (4)

- ❑ **Bibliothèque : MESCAN1\_Utilitaire.h**
- ❑ **bool MESCAN\_GetBit(uCAN1\_MSG \*CanMsg, MC\_Bit DefBit);**
- ❑ **uint8\_t MESCAN\_GetData8(uCAN1\_MSG \*CanMsg, MC\_Data8 DefData);**
- ❑ **void MESCAN\_SetData8(uCAN1\_MSG \*CanMsg, MC\_Data8 DefData, uint8\_t Data);**
- ❑ **void MESCAN\_SetData16(uCAN1\_MSG \*CanMsg, MC\_Data16 DefData16, uint16\_t Data16);**
- ❑ **uint16\_t MESCAN\_GetData16(uCAN1\_MSG \*CanMsg, MC\_Data16 DefData16);**
- ❑ **void MESCAN\_SetData32(uCAN1\_MSG \*CanMsg, MC\_Data32 DefData32, uint32\_t Data32);**
- ❑ **uint32\_t MESCAN\_GetData32(uCAN1\_MSG \*CanMsg, MC\_Data32 DefData32);**

## □ Spécification des niveaux de priorité d'une trame CAN, 0(bas) -> 3(elevee)

```
typedef enum
{
    CANLINUX_PRIORITY_HIGH = 0b11,
    CANLINUX_PRIORITY_MEDIUM = 0b10,
    CANLINUX_PRIORITY_LOW = 0b01,
    CANLINUX_PRIORITY_NONE = 0b00
} CANLINUX_TX_PRIORITY;
```

```
typedef struct CanBufferPrio
{
    iFILE_MesCAN fileHigh;
    iFILE_MesCAN fileMedium;
    iFILE_MesCAN fileLow;
    iFILE_MesCAN fileNone;
    int canPort;
    bool running;
} CanBufferPrio;
```

Files d'attente associée aux messages CAN

□ **int canLinux\_init\_prio (char \*portName);**

- ◆ Initialisation du CAN ainsi que les fifos de transmission. Enregistre également en interne le port CAN.
- ◆ retourne l'index du bus a utiliser par la suite pour des operations sur le socket.

□ **Exemple :**

```
char *NomPort = "can0";  
int canPort;  
canPort = canLinux_init_prio(NomPort)
```



# Commande train sur UniRAIL (1) : API canLinux



❑ **void canLinux\_close ();**

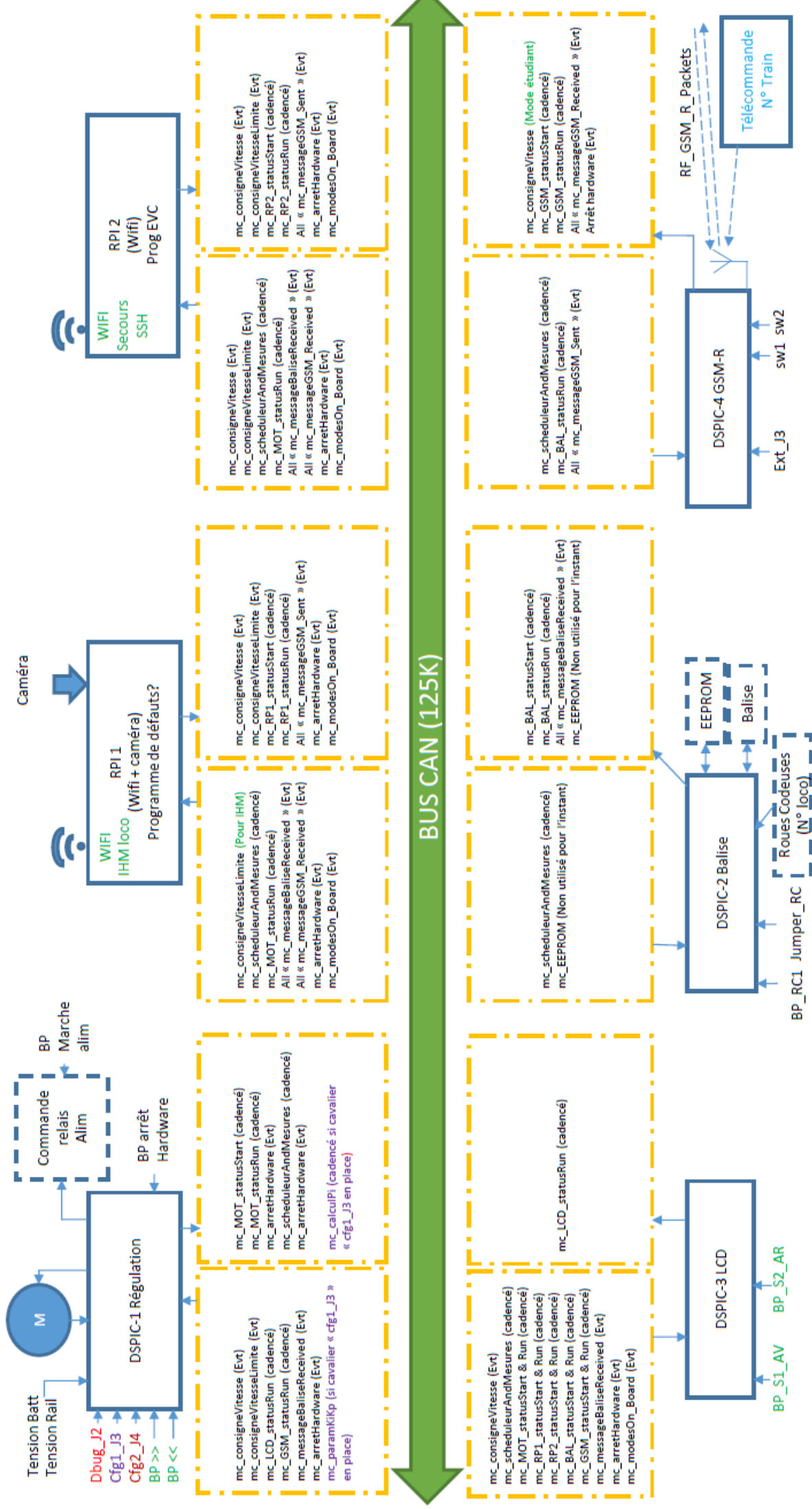
◆ Ferme l'accès aux FIFO pour l'envoi des nouveaux messages, attend qu'elles soient vides, puis détruit le thread d'envoi et ferme le port.

# Commande train sur UniRAIL (1) : Standard dénomination des variables

- MC : Message CAN pour constantes, mc: Message CAN pour variables.
- On se définit 6 diminutifs correspondant aux 6 processeurs d'une locomotive:
  - MOT: Carte régulation vitesse moteur. (Indice 0)
  - RP1 et RP2: Carte Rpi 1 et 2. (Indice 1 et 2)
  - BAL: Carte qui gère la communication avec deux balises. (Indice 3)
  - LCD: Carte affichage. (Indice 4)
  - GSM: Carte de communication Xbee. (Indice 5)
- Chaque variable identifiant un message CAN commencera par « mc » soit (message CAN), il sera suivi par un des 6 diminutifs ci-dessous si toutes ses datas sont gérées par cette même carte, ou si risque de confusion.
- Les datas de ces trames CAN peuvent être de différents formats: (bit, char, word, long ou float). De même que pour les noms des messages CAN, on peut insérer dans le nom des variables, un de ces 6 diminutifs si on a un risque de confusion.
- Pour les identifier, le nom de chaque variable sera codé de cette manière :
  - bdmc\_NomVariable : « bit » data message CAN
  - cdmc\_NomVariable : « char » data message CAN
  - wdmc\_NomVariable : « word » data message CAN
  - ldmc\_NomVariable : « long » data message CAN
  - float\_NomVariable : « float » data message CAN



# V3.1



- ❑ Structure de données pour définir un filtre :

```
struct can_filter {  
    canid_t can_id;  
    canid_t can_mask; };
```

- ❑ **void canLinux\_initFilter (struct can\_filter\* rfilter, int longueur);**

- ◆ Met en place le filtre

- ❑ Exemple :

```
struct can_filter rfilter[1]; //Le filtre sera limite ici a une variable  
rfilter[0].can_id = MC_ID_SCHEDULEUR_MESURES;  
rfilter[0].can_mask = CAN_SFF_MASK;  
canLinux_initFilter(rfilter, sizeof(rfilter));
```

- ❑ **bool canLinux\_transmit(CANLINUX\_TX\_PRIORITY priority, uCAN1\_MSG \*sendCanMsg);**
  - ◆ **Ecriture de données sur le bus CAN avec spécification du niveau de priorité**
  - ◆ **Retourne true si le message a été envoyé et false sinon**
- ❑ **Remarque : Ne traite qu'une seule trame CAN a la fois .**
- ❑ **Exemple :**

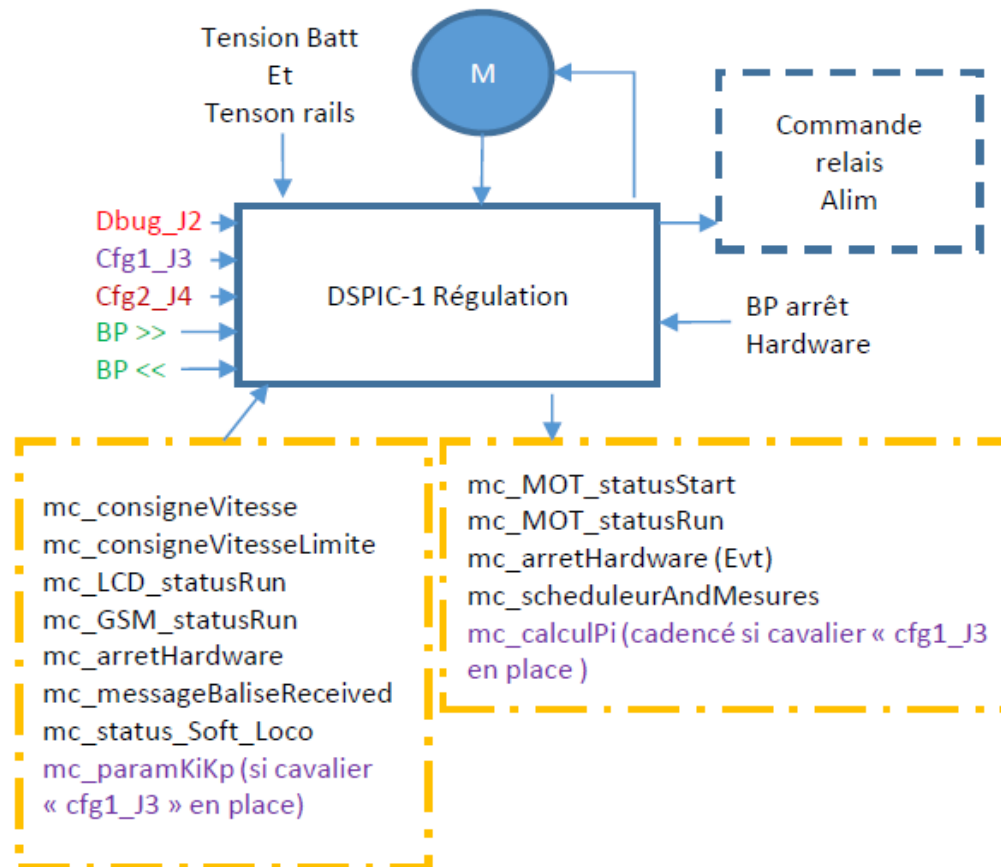
```
uCAN1_MSG consigneVitesse;  
consigneVitesse.frame.id = MC_ID_CONSIGNE_VITESSE;  
consigneVitesse.frame.dlc = MC_DLC_CONSIGNE_VITESSE;  
MESCAN_SetData8(&consigneVitesse, cdmc_consigneVitesse, vitesse);  
MESCAN_SetBit(&consigneVitesse, bdmc_sensDeplacementLoco, sense);  
canLinux_transmit(CANLINUX_PRIORITY_HIGH, &consigneVitesse);
```

- ❑ **bool canLinux\_receive(uCAN1\_MSG \*recCanMsg, unsigned int timeOut);**
  - ◆ Lecture de données circulant sur le bus CAN avec spécification d'un temps d'attente.
  - ◆ Retourne true si une trame CAN a été lue pendant la durée du temps d'attente, false sinon.
- ❑ **Remarque : Ne traite qu'une seule trame CAN a la fois .**
- ❑ **Exemple :**

```
uCAN1_MSG buffLectCAN;  
canLinux_receive(& buffLectCAN, 1);
```

- **bool messageIsGSM\_R\_SENT(int IDunderTest);**
  - ◆ Identifie si la trame CAN sélectionnée est de type GSMR\_SENT.
  - ◆ Retourne true si l'identifiant est du type GSMR\_SENT.
  - ◆ L'id des trames GSMR\_SENT est compris entre 0x60 - 0x6F
- **bool messageIsGSM\_R\_RECEIVED(int IDunderTest);**
  - ◆ Identifie si la trame CAN sélectionnée est de type GSMR\_RECEIVED.
  - ◆ Retourne true si l'identifiant est du type GSMR\_RECEIVED.
  - ◆ L'id des trames GSMR\_RECEIVED est compris entre 0x70 - 0x7F
- **bool messageIsEBT\_L2(int IDunderTest);**
  - ◆ Identifie si la trame CAN sélectionnée est de type EBT.
  - ◆ Retourne true si l'identifiant est du type EBT.
  - ◆ L'id des trames de type EBT est compris entre 0x30 - 0x3F.

# Messages CAN pour la commande du moteur



# Message CAN mc\_consigneVitesse limite (1)

MC\_ID\_CONSIGNE\_VITESSE LIMITE

MC\_DLC\_CONSIGNE\_VITESSE LIMITE

mc\_consigneVitesseLimite

↓ IN

ID	DLC	D[0]	D[1]	D[2]	D[3]	D[4]	D[5]	D[6]	D[7]
16	1	xx							

D[0] : cdmc\_consigneVitesseLimite  
la vitesse en cm/s ( 0 <= consigne <= 50 ou 0x32)

## Message CAN mc\_consigneVitesse limite (2)

```
int WriteVitesseLimite(float vitesseLimite)
{
    uCAN1_MSG consigneUrgence;

    if(vitesseLimite > MAX_CONSIGNE_VITESSE_AUTORISEE) //vitesse supérieur à 50 cm/s
        vitesseLimite = MAX_CONSIGNE_VITESSE_AUTORISEE;

    consigneUrgence.frame.id = MC_ID_CONSIGNE_VITESSE_LIMITE;
    consigneUrgence.frame.dlc = MC_DLC_CONSIGNE_VITESSE_LIMITE;
    MESCAN_SetData8(&consigneUrgence, cdmc_consigneVitesseLimite, vitesseLimite);

    return    canLinux_transmit(CANLINUX_PRIORITY_HIGH, &consigneUrgence);
}
```



# Message CAN mc\_consigneVitesse (1)

MC\_ID\_CONSIGNE\_VITESSE

MC\_DLC\_CONSIGNE\_VITESSE

mc\_consigneVitesse

↓ IN

ID	DLC	D[0]	D[1]	D[2]	D[3]	D[4]	D[5]	D[6]	D[7]
17	2	xx	xx						

D[0] : cdmc\_consigneVitesse

la vitesse en cm/s ( 0 <= consigne <= 50 ou 0x32) →Ecrêtage

D[1] :

bit0: bdmc\_sensDeplacementLoco

le sens (bit0 = 1 -> sens Avant, bit0 = 0 -> sens Arrière)

bit1: bdmc\_ForcageMvtLoco (Télécommande ou boutons « Wagon »)

NB : 16,944ms équivaut à 6,78mm à 40cm/s

## Message CAN mc\_consigneVitesse (2)

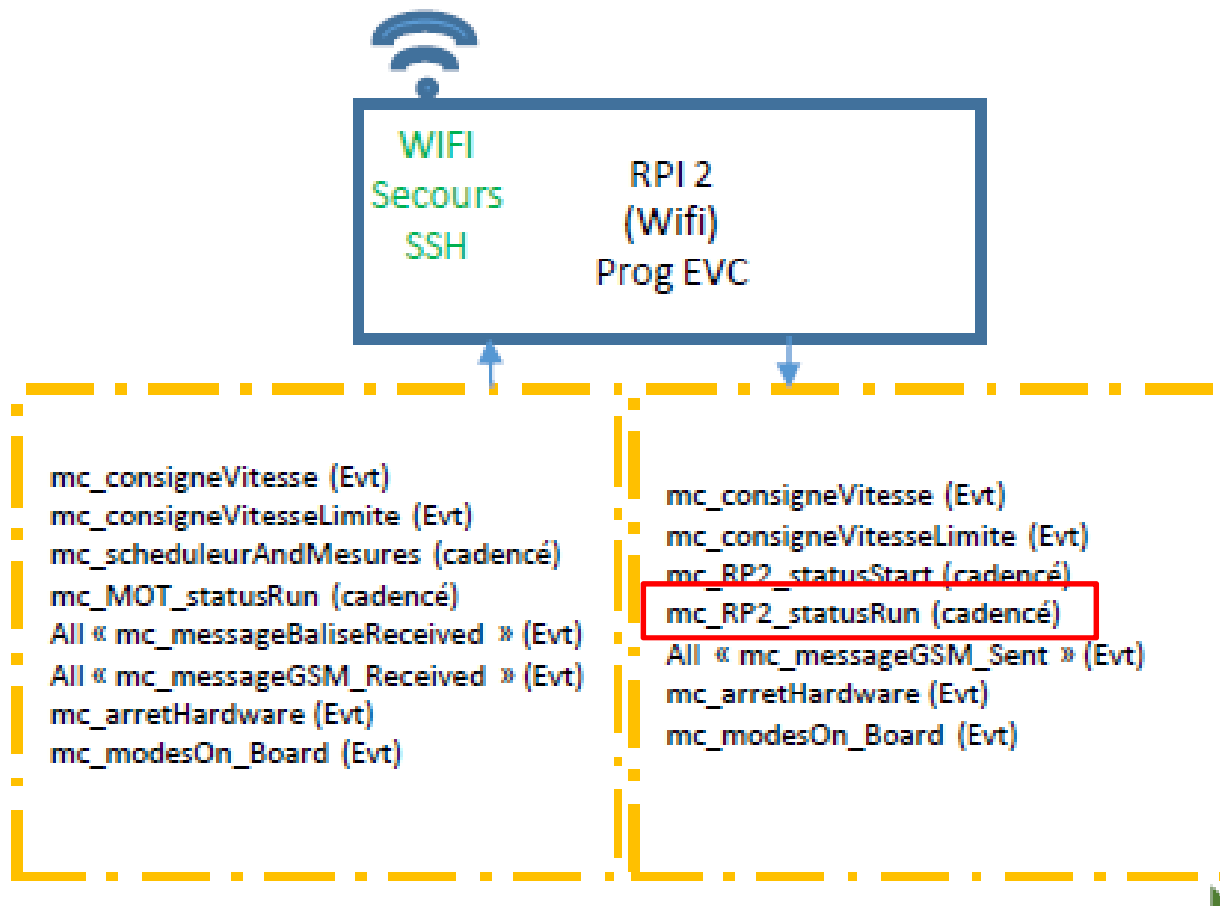
```
int WriteVitesseConsigne(unsigned int vitesse, unsigned char sense)
{
    uCAN1_MSG consigneVitesse;

    if(vitesse>MAX_CONSIGNE_VITESSE_AUTORISEE) //vitesse supérieur à 50 cm/s
        vitesse = MAX_CONSIGNE_VITESSE_AUTORISEE;

    consigneVitesse.frame.id = MC_ID_CONSIGNE_VITESSE;
    consigneVitesse.frame.dlc = MC_DLC_CONSIGNE_VITESSE;
    MESCAN_SetData8(&consigneVitesse, cdmc_consigneVitesse, vitesse);
    MESCAN_SetBit(&consigneVitesse, bdmc_sensDeplacementLoco, sense);

    return    canLinux_transmit(CANLINUX_PRIORITY_HIGH, &consigneVitesse);
}
```

# Message CAN du raspberry PI1-EVC (1)



# Message CAN du raspberry PI1-EVC (1)

Nombre d'impulsions entre deux périodes

**mc\_schaleurAndMesures** (Envoyée toutes les 16,944ms ce qui équivaut à 6,78mm à 40cm/s)

	ID	DLC	D[0]	D[1]	D[2]	D[3]	D[4]	D[5]	D[6]	D[7]
↑ OUT	2F	8	xx	xx	xx	xx	xx	xx	xx	xx

D[0] : cdmc\_ordonnancementId. Identifiant trames status (ordonnancement et synchronisation)

D[1] : cdmc\_vitesseMesurée

D[2],D[3] : wdmc\_QEI\_distanceRelativeParcourue (PS: sera RAZ quand réception TrameBalise1)

D[4],D[5] : wdmc\_valeurErreurRegul

D[6] : cdmc\_vitesseConsigneInterne

D[7] : Reserved

NB QEI: Quadrature Encoder Interface (Périphérique uC)

# Message CAN du raspberry PI1-EVC (1)

```
int WriteTrameStatusRUNRP1(unsigned char status, unsigned char varDebug1, unsigned char varDebug2)
{
    uCAN1_MSG trameStatusRP1;
    trameStatusRP1.frame.id = MC_ID_RP1_STATUS_RUN;
    trameStatusRP1.frame.dlc = MC_DLC_RP1_STATUS_RUN;
    MESCAN_SetData8(&trameStatusRP1, cdmc_RP1_erreurs, 0);
    MESCAN_SetData8(&trameStatusRP1, cdmc_RP1_warnings, 0);
    MESCAN_SetBit(&trameStatusRP1, bdmc_RP1_etatConnexionWIFI, 1);
    MESCAN_SetData8(&trameStatusRP1, cdmc_RP1_configONBOARD, status);
    MESCAN_SetData8(&trameStatusRP1, cdmc_RP1_var1Debug, varDebug1);
    MESCAN_SetData8(&trameStatusRP1, cdmc_RP1_var2Debug, varDebug2);

    return    canLinux_transmit(CANLINUX_PRIORITY_MEDIUM, &trameStatusRP1);
}
```



**Fin**